# Lecture 8 - AI subsystem

*"A computer lets you make more mistakes faster than any invention in human history." - Mitch Radcliffe*

- Pathfinding
- Behaviour

## Last Week Recap

- Graphics and Audio

## Pathfinding

- Used by AI agents (entities) to calculate a valid route to player (entity)
- Usually game world is converted to grid / graph data structure
- A* algorithm is popular for its performance and result

## Graphs

- A collection of nodes and edges
- Nodes represent rooms / tiles / a single movement block
- Edges represent the existence of a path between 2 rooms / tiles / blocks

## Game World -> Graph

```
let graph = new Graph();

world.tiles.forEach(tile => graph.addNode(node(tile)));

graph.nodes.forEachPair(node1, node2 =>
    if (node1.tile can go to node2.tile) {
        graph.addEdge(node1, node2);
    }
);
```

The result is a *connected* graph, assuming you can get to any tile from any other tile.

## Apply Algorithms

We have converted a specific problem to a generic problem. We can now apply existing algorithms to solve our problem.

## Apply A*

```
auto path = AStar.apply(graph, startTile, endTile);
```

`path` contains a list of tiles, which is the shortest route from `startTile` to `endTile`.

## Activity

Suppose we have a maze (for example the one in the assignment code). How would you solve the problem of finding a path from the start to the end of the maze?

**Behaviour**

The aim is to make AI controlled entities just smart enough to make them engaging and just dumb enough to make the player seem better (smarter, stronger, etc.).

## Behaviour Perception

The key property of the behaviour is to be consistent.
Inconsistencies lead to player frustration.

## Behaviour Types

- Movement
- Decision Making (what to do now?)
- (Group) Strategy (what to do next?)

## Various AI Behaviour Implementation Types

- FSM (Finite State Machine)
- BTrees (Behaviour Trees)
- GOAP (Goal Oriented Action Planning)
- Squad Tactics

## FSM

- Can only be in 1 state at a time
- Works for most simple AI and some complex AI

## FSM Impl

```
stateMachine.currentState.update();

// state change
stateMachine.currentState = State.RUN;
stateMachine.currentState = State.WALK;
```

Each state has its own `update()` function to implement.

## Activity

Design and implement (on paper) a two-state FSM for your player.
Examples: walk / fly, run / swim, active / passive.

## BTrees

- Can define complex behaviour from simple actions
- It's like the ECS pattern of AI (can mix and match tree elements)
- Flexible

## BTrees Theory

1. Start with simple (possibly atomic) actions: move, shoot, throw grenade, pick up weapon, etc.
2. Identify the order in which to complete them.
3. See if there are any conditions associated with an action. For example, to shoot an entity, the entity must be in sight.

## Simple Example

```
root
    move
    shoot
    move
    throw grenade
```

## More Complex Example

```
root
    selector
        sequence
            move
            shoot
        sequence
            move
            throw grenade
```

The AI tries to move and shoot, if it fails (e.g. no ammo), then tries to move and throw a grenade.

## Pac-man Demo

Let's examine some example btrees in a pac-man demo.

## GOAP (F.E.A.R, FF XV)

- Define actions (like BTree) but do not specify order, i.e. dynamic
- Use game world queries to determine its state
- Build a chain of actions that will lead AI to the “best” world state for AI
- FF XV AI systems video

## GOAP Example

```
Action: Shoot Player
Precondition: Ammo in weapon, player visible
Postcondition: Player dead

Action: Pick Up Ammo
Precondition: no ammo
Postcondition: Ammo in weapon
```

## GOAP Theory

1. Convert each action into a graph node
2. Add edges between nodes if postcondition matches precondition
3. Compute shortest path from initial state to goal state
4. (Optional) Can add weights to edges to make the AI seem more effective

## Squad AI

- AI seem to communicate between units
- Much more dynamic / non-deterministic experience
- Squad “center of mass”

## Squad AI Theory

Suppose a squad is being fired at.

1. Identify valid positions for cover (search space).
2. Add each position a weight based on threat level.
3. Compute `n` "best" positions, where `n` is number of AI units.
4. Assign each position to a unique unit.
5. (Optional) Can give better positions to more valuable units.

## AI Behaviour

- Can be implemented via `AIComponent`
- Different per each game context (e.g. Fallout 4 vs Assassin's Creed)

## Activity

1. Think of an existing game, such as Fallout 4 and pick an AI type.
2. What is the aim of the AI you picked?
3. Design the AI you picked using one of the behaviour types above.

## Conclusion

- Many AI algorithms already exist
- Find a way to translate your problem to one that's been solved
- Important to "contain" AI, as it can overwhelm the player

## Extra Reading

- Good explanation of btrees - gdxAI
- Combining Design and AI - GDC Alien Isolation